C# 프로그래밍

- 6 주차 (1강)

SEJONG UNIVERSITY

객체지향 프로그래밍

- 은닉성 (캡슐화)
- 상속성
- 다형성

Introduction to 다형성 (Polymorphism)

- 부모 클래스 객체가 자식 클래스 메소드를 사용하려면, 자식 클래스로 형식 변환 필요
  - ✓ 조건문 통해 자식 클래스 타입을 구별하고 해당 타입의 메소드 호출

```
class Animal
{
    public int Age { get; set; }
    public Animal() { this.Age = 0; }

    public void Eat() { Console.WriteLine("냠냠 먹습니다."); }
    public void Sleep() { Console.WriteLine("쿨쿨 잠을 잡니다."); }
}

class Dog : Animal
{
    public string Color { get; set; }
    public void Bark() { Console.WriteLine("멍멍 짓습니다."); }
}

class Cat : Animal
{
    public void Meow() { Console.WriteLine("냥냥 웁니다."); }
}
```

- 다형성은 형식변환 수행 없이 자식 클래스 메소드 호출 지원

```
static void Main(string[] args)
{
    Animal[] Animals = new Animal[]{ new
Dog(), new Cat(), new Dog(), new Cat()};

    foreach (var item in Animals)
    {
        item.Eat();
        if(item is Dog)
            ((Dog)item).Bark();
        else
            ((Cat)item).Meow();
    }
}
```

출력 결과:

```
냠냠 먹습니다.
멍멍 짓습니다.
냠냠 먹습니다.
냥냥 웁니다.
냠냠 먹습니다.
멍멍 짓습니다.
냠냠 먹습니다.
냥냥 웁니다.
```

다형성 (Polymorphism)

- 클래스의 객체가 여러 형태를 가질 수 있음을 의미
  - ✓ 자신을 상속받아 만들어진 자식 클래스를 통해 다형성 실현
- 다형성 지원을 위한 핵심 기능은 메소드 오버라이드 (override)
  - ✓ 자식 클래스에서 부모 클래스의 메소드를 재정의
  - ✓ 부모 클래스 메소드에 virtual 키워드, 자식 클래스 메소드에 override 키워드 사용
  - ✓ 용도가 비슷해 보일 수 있는 메소드 하이딩 (hiding) 또는 오버로딩 (overloading) 과의 구분 필요
- 메소드 오버라이드와 오버로드 사이의 차이
  - ✓ 오버라이드의 경우 부모/자식 클래스 메소드의 이름, 반환타입, 매개변수의 수, 개별 매개변수 타입이 모두 같음
  - ✓ 오버로드는 매소드 이름은 같지만, 매개변수의 수 또는 개별 매개변수 타입이 다름

변수 하이딩 (hiding)

- 부모와 자식 클래스에 이름이 같은 프로퍼티 또는 필드 변수를 선언
  - ✓ new 키워드를 사용함
  - ✓ 자식 클래스의 객체는 자식 클래스의 변수, 부모 클래스의 객체는 부모 클래스의 변수 사용

```
class Parent
{
    public int Variable { get; set; } = 273;
}

class Child : Parent
{
    public new string Variable { get; set; } = "hiding";
}
```

```
class MainApp
{
    static void Main(string[] args)
    {
        Child child = new Child();
        Console.WriteLine(child.Variable);
        Console.WriteLine(((Parent)child).Variable);
    }
}
```

출력 결과:
hiding
273

메소드 하이딩 (hiding)

- 부모와 자식 클래스에 반환 값, 이름, 매개변수 모두 같은 메소드를 선언
  - ✓ New 키워드를 사용함
  - ✓ 자식 클래스 객체는 자식 클래스의 메소드, 부모 클래스 객체는 부모 클래스의 메소드 사용

```
class Parent
{
    public void Method()
    {
        Console.WriteLine("부모의 메소드");
    }
}

class Child : Parent
{
    public new void Method()
    {
        Console.WriteLine("자식의 메소드");
    }
}
```

```
class MainApp
{
    static void Main(string[] args)
    {
        Child child = new Child();
        child.Method();
        ((Parent)child).Method();
    }
}
```

출력 결과:
자식의 메소드
부모의 메소드

메소드 오버라이딩 (overriding)

- 자식 클래스에서 부모 클래스의 메소드를 재정의
- 부모와 자식 클래스에 반환 값, 이름, 매개변수 모두 같은 메소드를 선언
  - ✓ 부모 클래스 메소드에 virtual 키워드, 자식 클래스 메소드에 override 키워드 사용
  - ✓ 부모/자식 클래스의 객체 모두 자식 클래스에서 재정의한 메소드 사용
- base 키워드 이용해 부모 클래스 메소드 호출 가능
  - ✓ 자식 클래스 메소드 안에서 부모 클래스 메소드 호출
  - ✓ 부모 클래스의 메소드 안에 코드를 자식 클래스 안에서 중복해서 작성할 필요 없음

메소드 오버라이딩 예제 코드

```
class Animal
{
    public virtual void Cry() { Console.WriteLine("소리 내어 웁니다."); }
}

class Dog : Animal
{
    public override void Cry()
    {
        Console.WriteLine("멍멍 짓습니다.");
    }
}

class Cat : Animal
{
    public override void Cry()
    {
        base.Cry();
        Console.WriteLine("냥냥 웁니다.");
    }
}
```

```
static void Main(string[] args)
{
    Animal[] Animals = new Animal[] { new Dog(),
new Cat(), new Dog(), new Cat() };

    foreach (var item in Animals)
    {
        item.Cry();
    }
}
```

출력 결과:

```
멍멍 짓습니다.
소리 내어 웁니다.
냥냥 웁니다.
멍멍 짓습니다.
소리 내어 웁니다.
냥냥 웁니다.
```

메소드 하이딩과 출력 결과 비교

```
class Animal
{
    public virtual void Cry() { Console.WriteLine("소리 내어 웁니다."); }
}

class Dog : Animal
{
    public new void Cry() // 메소드 하이딩
    {
        Console.WriteLine("멍멍 짓습니다.");
    }
}

class Cat : Animal
{
    public new void Cry()  // 메소드 하이딩
    {
        base.Cry();
        Console.WriteLine("냥냥 웁니다.");
    }
}
```

```
static void Main(string[] args)
{
    Animal[] Animals = new Animal[] { new Dog(),
new Cat(), new Dog(), new Cat() };

    foreach (var item in Animals)
    {
        item.Cry();
    }
}
```

출력 결과:

```
소리 내어 웁니다.
소리 내어 웁니다.
소리 내어 웁니다.
소리 내어 웁니다.
```

오버라이딩 제한: sealed 키워드

- sealed 키워드를 메소드 앞에 붙이면 더 이상 오버라이딩하지 말라는 의미
  - ✓ 원래 virtual 키워드가 붙어 오버라이딩할 수 있는 메서드를 어느 지점 이후로 못하게 함

```
class Base {
    public virtual void SealMe() { }
}

class Derived : Base {
    public sealed override void SealMe() { }
}

class WantToOverride : Derived {    // 컴파일 에러
    public override void SealMe() { }
}
```

```
class MainApp
{
    static void Main(string[] args)
    {
    }
}
```

object 기본 메소드 확장

- 일반적으로 ToString 의 경우 클래스의 인스턴스 값을 적절하게 표현하도록 재정의

```
public class Object
{
    public virtual bool Equals();
    public virtual string ToString();
    // …
}
```

```
public class Point
{
    int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public override string ToString()
    {
        return "X: " + x + ", Y: " + y;
    }
}
```

```
class MainApp
{
    static void Main(string[] args)
    {
        Point pt = new Point(5, 10);
        Console.WriteLine(pt.ToString());
    }
}
```

출력 결과:
X: 5, Y: 10

object 기본 메소드 확장

- 동일한 책인지 확인하기 위해 ISBN 필드 정보를 비교하도록 Equals () 메소드 재정의

```
class Book
{
    decimal isbn13;
    string title;
    string contents;

    public Book (decimal isbn13, string title, string contents) {
        this.isbn13 = isbn13;
        this.title = title;
        this.contents = contents;
    }

    public override bool Equals(object obj) {
        if(obj == null) { return false; }

        Book book = obj as Book;
        if (book == null) { return false; }
        return this.isbn13 == book.isbn13;
    }
}
```

```
static void Main(string[] args)
{
    Book book1 = new Book(1234, "book1", "...");
    Book book2 = new Book(1234, "book1", "...");
    Book book3 = new Book(5678, "book3", "...");

    Console.WriteLine("book1 == book2: " + book1.Equals(book2));
    Console.WriteLine("book1 == book3: " + book1.Equals(book3));
}
```

출력 결과:

```
book1 == book2: True
book1 == book3: False
```

메소드 오버로드

- 메소드의 이름은 같지만, 매개변수의 수 또는 매개변수의 타입이 다른 다수의 메소드 정의
  - ✓ 메소드의 반환 타입은 고려하지 않음

```
class Mathematics
{
    public int Abs (int value)
    {
        return (value >= 0) ? value : -value;
    }

    public double Abs(double value)
    {
        return (value >= 0) ? value : -value;
    }

    public decimal Abs(decimal value)
    {
        return (value >= 0) ? value : -value;
    }
}
```

```
static void Main(string[] args)
{
    Mathematics math = new Mathematics();
    Console.WriteLine(math.Abs(-5));
    Console.WriteLine(math.Abs(-10.052));
    Console.WriteLine(math.Abs(20.01m));
}
```

출력 결과:
5
10.052
20.01

연산자 오버로드

- 연산자를 타입 별로 재정의 하여 사용
  - ✓ 예를 들어, + (더하기) 연산자를 보면 정수형 타입과 문자열 타입에 연산자 역할이 다름
  - ✓ 정수형 타입에서는 정수의 덧셈 역할
  - ✓ 문자열 타입에서는 문자열을 이어 붙이는 역할

```
static void Main(string[] args)
{
    int n1 = 5;
    int n2 = 10;
    int sum = n1 + n2;  // sum 값은 15

    string txt1 = "123";
    string txt2 = "456";
    Console.WriteLine(txt1 + txt2); // 출력 결과 123456
}
```

사용자 정의 타입 : 연산자 오버로드 없는 경우

- 연산자 오버로드 없이 더하기 연산은 일반적인 메소드 이용해 각 기능 구현
  - ✓ 예를 들어, 무게의 단위를 나타내는 Kilogram 을 클래스로 정의

```
public class Kilogram
{
    double mass;
    public Kilogram (double value)
    {
        this.mass = value;
    }

    public Kilogram Add (Kilogram target)
    {
        return new Kilogram(this.mass + target.mass);
    }

    public override string ToString()
    {
        return mass + "kg";
    }
}
```

```
static void Main(string[] args)
{
    Kilogram kg1 = new Kilogram(5);
    Kilogram kg2 = new Kilogram(10);

    Kilogram kg3 = kg1.Add(kg2);
    Console.WriteLine(kg3); // 출력 결과: 15 kg
}
```

사용자 정의 타입 : 연산자 오버로드 사용

- 연산자 오버로드를 이용하여 + 연산자에 의미 부여

```
public class Kilogram
{
    double mass;
    public Kilogram (double value)
    {
        this.mass = value;
    }

    public static Kilogram operator +(Kilogram op1, Kilogram op2)
    {
        return new Kilogram(op1.mass + op2.mass);
    }

    // ... [생략] ...
}
```

사용형식

```
public static 타입 operator 연산자 (타입1 변수명1, 타입2 변수명2)
{
    // [타입]을 반환하는 코드
}
```

```
static void Main(string[] args)
{
    Kilogram kg1 = new Kilogram(5);
    Kilogram kg2 = new Kilogram(10);

    Kilogram kg3 = kg1 + kg2;
    Console.WriteLine(kg3); // 출력 결과: 15 kg
}
```

Thank you!